Об алгоритмизации знаний

А. С. Подколзин[1]

В работе рассматриваются вопросы компьютерного моделирования процессов решения задач и автоматического создания приемов решателя. Излагается архитектура процесса перехода от теорем к приемам (алгоритмизации знаний), сложившаяся в процессе обработки многочисленных примеров. Создана компьютерная система, позволяющая решать задачи из различных разделов математики и способная пополнять свою базу приемов.

Ключевые слова: компьютерный решатель задач, логическая система, искусственный интеллект.

В компьютерной сети хранятся огромные запасы знаний, накопленных человечеством. Казалось бы, отсюда рукой подать до искусственного интеллекта, способного применять эти знания для решения разнообразных задач и пополнять их запасы. Однако, время идет, а программы по-прежнему пишутся программистами, науки развиваются учеными, технические проекты создаются инженерами. Всплеск энтузиазма, вызванного успехами искусственных нейросетей в распознавании образов, этой картины не изменил. Создается такое впечатление, что центральной проблемой искусственного интеллекта является, все-таки, не проблема распознавания образов, а проблема алгоритмизации знаний. Нужно научить компьютер самостоятельно создавать по хранящимся в нем знаниям алгоритмы/приемы решения задач, и лишь тогда возникнет искусственный интеллект, по-настоящему понимающий эти знания.

С точки зрения математики, проблема алгоритмизации знаний сводится к вопросу: как из теорем извлечь приемы решения задач? Для изучения процесса преобразования теорем в приемы и была создана компьютерная система, описанию которой посвящена монография "Компьютерное моделирование логических процессов". Эту монографию [1], а также программу самой системы можно найти на сайте кафедры МаТИС www.intsys.msu.ru.

Прежде всего, пришлось накопить достаточно большое количество уже "алгоритмизированных" знаний и понять, как они должны быть организованы в компьютерной системе для того, чтобы она могла эффективно решать задачи. Прорабатывались примеры из множества различных предметных областей. Процесс решения разбивался на элемен-

[1] *Подколзин Александр Сергеевич* — д.ф.м.н., профессор каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: alexander.p@yandex.ru.

Podkolzin Alexander Sergeevich — Dr. of Sc., Professor, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

тарные шаги, и для каждого из них предлагалось объяснение в виде небольшой программы — "приема который в аналогичной ситуации выполнял аналогичные действия. Для приема определялось то ключевое понятие, появление которого в текущем контексте должно было инициировать попытку применения приема. За каждым понятием закреплялась ветвь программы решателя, к которой относились соответствующие приемы, и база приемов оказалась организована как энциклопедия приемов. Решение задачи происходило в процессе сканирования ее описания и обращения для текущего понятия к соответствующей ветви данной энциклопедии. Таким образом, система получила что-то вроде внутреннего "логического зрения" и могла принимать решение об очередном действии с учетом всей текущей картины.

Так как каждый прием действовал автономно и независимо от других приемов, а общее количество приемов на текущий момент достигло более чем 50000, возникла проблема организации разумного их взаимодействия, которое закладывалось в решающие правила приемов в процессе обучения на примерах. Всего было рассмотрено более 13000 задач из различных предметных областей: дискретная математика, алгебра множеств, элементарная алгебра, элементарная геометрия, аналитическая геометрия, линейная алгебра, математический анализ, дифференциальные уравнения, интегральные уравнения, комплексный анализ, теория вероятностей, общая алгебра, элементарная физика, элементарная химия, распознавание рукописных букв, текстовый анализ, шахматы. Система отображает процесс решения задачи "по шагам" и оказалась способна решать многие задачи уровня конкурсных экзаменов по математике. Например, пошаговый показ решения задач по элементарной алгебре, который в последнее время демонстрирует программа С.Вольфрама, данная система умела делать еще 25 лет назад. Неплохо справляется она со стандартными задачами по элементарной геометрии, а также задачами из других перечисленных выше разделов. Таким образом, можно считать, что понимание того, как выглядят алгоритмизированные знания, было до некоторой степени достигнуто.

Чтобы упростить создание приемов, был создан специальный язык программирования ЛОС (Логический Описатель Ситуаций), максимально приближенный к логическому языку. Главной его задачей была формулировка сложных условий на целесообразность применения приема в текущем контексте. В этих условиях разрешалось использовать кванторы и описатели, причем операторы языка работали в режиме перечисления значений выходных переменных, позволившем обходиться без операторов цикла. Хотя язык ЛОС и оказался близок к известному языку логического программирования ПРОЛОГ, в отличие от ПРОЛОГа он ориентирован не на формулировку теоремы предметной области, а на

формулировку условий управления этой теоремой. Для решателей это оказалось более важным, так как часто запись управляющей компоненты приема была во много раз сложнее записи теоретической компоненты. ЛОС существенно ускорил процесс программирования и упростил чтение программ. Для выполнения его программ создан интерпретатор, и вся работа системы, включая интерфейсы, происходит через ЛОС.

Дальше начался процесс постепенного движения вспять — от алгоритмизированных в виде приемов знаний к их источникам. Практика обучения решателя показала, что обычно прием основан на какой-то единственной теореме. В программе ЛОСа фрагменты этой теоремы и управления теоремой перемешаны достаточно хаотичным образом. Естественным первым шагом на пути к истокам программ стал переход к раздельной записи теоремы и управления. Для такого разделения был создан язык логического программирования ГЕНОЛОГ, в котором прием задается теоремой, сопровожденной некоторой алгоритмизирующей разметкой ("генотипом" приема), понятной компилятору. Этот язык, в отличие от ЛОСа, не является непосредственно исполняемым. Компилятор преобразует описание приема на ГЕНОЛОГе в программу ЛОСа.

Чтобы сформулировать условия целесообразности применения теоремы в текущем контексте, ГЕНОЛОГ использует полномасштабный логический язык. Таким образом, описание приема имеет два логических уровня — уровень предметной области, на котором задается теорема, и уровень структур данных, на котором задается управление теоремой. В этом заключается принципиальное отличие ГЕНОЛОГа от других языков логического программирования. Создание ГЕНОЛОГа происходило постепенно, по мере проработки задач из различных разделов. Фактически, он представляет собой огромную коллекцию способов алгоритмизации теорем. ГЕНОЛОГ настолько упростил и ускорил создание приемов, что позволил в сравнительно короткие сроки накопить их запас, достаточный, например, для решения задач по планиметрии. Проработка перечисленных выше разделов, в которых было создано более 50000 приемов решателя, была осуществлена на ГЕНОЛОГе. Первые шесть томов монографии [1, 2, 3, 4, 5, 6] "Компьютерное моделирование логических процессов" посвящены изложению этих приемов и описанию общей организации компьютерной логической системы.

Однако, ГЕНОЛОГ оказался лишь промежуточным пунктом на пути от приемов к породившим их теоремам. Алгоритмизирующая разметка теоремы была нацелена лишь на то, чтобы подробно объяснить компилятору, как по теореме создавать ЛОС-программу приема. В ней ничего не говорилось о целях применения приема. Чтобы автоматизировать создание таких разметок, была предпринята классификация приемов ГЕНОЛОГа по целевому признаку. Согласно этой классификации, целе-

вая установка приема описывалась типом приема и небольшим набором сопровождающих его данных. Например, направлением тождественной либо эквивалентной замены, выделением каких-то переменных, подтермов, и т.п. Такая целевая установка, получившая название спецификации приема, оказалась фактически альтернативным способом задания приема. Язык задания приемов с помощью сопровождающих теорему спецификаций был назван логическим ассемблером. Аналогия с обычным ассемблером, хотя и весьма отдаленная, заключается в том, что тип приема уподобляется коду операции, а дополнения к нему — операндам.

Число типов приемов приближается к 1500. Наиболее часто встречаются порядка 300 из них. Для перехода от задания приема на логическом ассемблере к заданию его на ГЕНОЛОГе был создан компилятор. Однако, этот процесс компиляции потребовал привлечь принципиально новый элемент — доводку создаваемого приема на задачах с целью оптимизации его параметров и бесконфликтного "вживления" в базу приемов.

Логический ассемблер, хотя и оказался языком пограничного слоя между теоремами и программами, примыкает к этому слою со стороны программ. В первую очередь, из-за того, что теоремы приемов оказалось целесообразно, в целях упрощения компиляции, несколько "деформировать" по отношению к обычным теоремам, отбрасывая избыточные проверки и добавляя некоторые элементы технического характера. Все-таки, теоремы приемов представляют собой лишь фрагмент языка программирования. Чтобы перейти через "пограничный слой" между приемами и теоремами и далее продолжить работу со стороны базы теорем, нужно было прежде всего создать эту самую базу теорем.

Заполнение базы теорем непосредственно из учебников привело бы к существенному разрыву между ними и теоремами приемов. Теоремы приемов обычно содержали множество обобщающих параметров или представляли собой какие-то комбинации теорем "из учебников ориентированные на решение задач и в учебниках обычно отсутствующие. Чтобы проследить источники приемов, нужно было избежать указанного разрыва. Поэтому первоначально база теорем заполнялась теоремами, представляющими собой аккуратные с точки зрения логики переформулировки теорем приемов. Она представляла собой как бы "проекцию" базы приемов. В большинстве случаев теорема из базы теорем попросту совпадала с теоремой приема.

Следующим вопросом было: как по теореме создавать спецификации приемов? Имеющаяся база теорем, привязанная к базе приемов и к уже готовым их спецификациям, позволила провести определенную классификацию теорем и выработать некоторый список стандартных характеристик теорем, подсказывающих возможные типы приемов для них. Большинство этих характеристик легко вычислялись непосредственно

по теореме, и для сопровождения ими теоремы была создана специальная процедура, названная характеризатором. Создание других характеристик требовало понимания предыстории возникновения теоремы. Они должны были появляться лишь в процессе вывода теорем. Так или иначе, в базе теорем каждая теорема сопровождалась списком своих характеристик. Спецификации приемов создавались процедурой, просматривающей характеристики теоремы и предлагающей для текущей характеристики список возможных спецификаций. Эта процедура получила название спецификатора.

Собственно говоря, уже с этого момента появилась возможность автоматического создания приемов по теореме: сначала характеризатор сопровождает теорему списком характеристик, затем спецификатор предлагает по каждой из них возможные спецификации, далее компилятор спецификаций преобразует их в описания приемов ГЕНОЛОГа, и, наконец, компилятор ГЕНОЛОГа получает ЛОС-программы приемов.

Однако, такие приемы, созданные без учета того, какие приемы уже имеются в решателе, обычно оказываются бесполезными или даже вредными. Либо они дублируют то, что делалось другими приемами, либо бесплодные попытки их применения сильно замедляют работу, либо они вообще направляют ход решения по ошибочному руслу. Чтобы преодолеть это явление, понадобились еще два этапа обработки приема.

Прежде всего, предпринимается попытка создать для приема простую тестовую задачу, которая решалась бы данным приемом, но не решалась в его отсутствие. Так как тип приема известен и известна его целевая ориентация, достаточно, чтобы задача была лишь одноходовкой, проверяющей, что решатель способен сделать шаг в направлении нужной цели. Данный этап обеспечивает настолько хорошую фильтрацию, что ее проходят только те приемы, которые действительно расширяют возможности решателя. Фактически, тестовый пример служит как бы доказательством необходимости приема.

Однако, даже необходимый для одной задачи прием бывает способен "поломать" ход решения других задач обучающего материала, сохраняемого в задачнике решателя. Поэтому, после примерки на тестовых задачах, предпринимается прокрутка решателя по одному или нескольким разделам задачника для выявления тех задач, решение которых сильно замедлилось или на которые стал возникать отказ. На этих задачах предпринимается доводка приема: варьируется уровень срабатывания приема, предпринимается переход к подтипу приема, обеспечивающему более высокую степень мотивированности срабатывания, и т.п. При доводке учитывается, что прием по-прежнему должен решать свою тестовую задачу.

Лишь после примерки и доводки автоматически созданные приемы регистрируются в накопителе результатов. Так как система находится лишь на стадии обучения, окончательный отбор приемов из накопителя и перенесение их в основную базу приемов пока выполняется вручную. Обычно отклоняется лишь меньшая их часть, причем причиной служит крайне маловероятное возникновение ситуации, на которую рассчитан прием.

Но вернемся к рассмотрению теорем — до того момента, как для них генерировались спецификации. Как уже говорилось, те теоремы, по которым создаются приемы, редко совпадают с "базисными" теоремами из учебников. Обычно они представляют собой результат определенной переработки базисных теорем, необходимой для решения задач. Такая переработка может заключаться в том, что теорема снабжается множеством обобщающих параметров, ориентированных на применение ее в "неявных" ситуациях, либо в комбинировании нескольких теорем для вывода стандартной "заготовки" для часто встречающейся в задачах ситуации, и т.п. Поэтому, для завершения рассмотрения цикла алгоритмизации теорем, осталось обеспечить указанный переход от базисных теорем к теоремам, по которым будут создаваться приемы. Этот переход будем называть программирующим логическим выводом.

Извлеченные из базы приемов решателя теоремы оказались превосходным обучающим материалом для создания приемов программирующего вывода. Они были распределены по специальным подразделам оглавления базы теорем — своего рода задачам на программирующий вывод. В первом пункте подраздела располагались одна или несколько базисных теорем, в остальных пунктах размещались те теоремы — источники приемов, которые должны были получаться программирующим выводом из базисных теорем. Эти подразделы получили название ячеек логического вывода.

Разумеется, доказательства теорем изложены в учебниках и хорошо известны. Не составляет особого труда и доказательство их следствий, используемых для создания приемов. Но умение доказывать теоремы ничего не дает, если сами теоремы еще отсутствуют. Поэтому проработка программирующего логического вывода означала ни много ни мало анализ процессов "открытия" теорем, начиная хотя бы с их простых следствий.

Чтобы объяснять, как та или иная теорема ячейки могла бы быть открыта при анализе базисных теорем, приходилось находить цепочку достаточно естественных переходов, быть может с привлечением дополнительных теорем, которые тоже заносились в общую базу теорем — для дальнейшего объяснения их "происхождения". Для устанавления того, какие переходы являются естественными, использовались характеристи-

ки теорем. Они позволили придать переходам в цепочке вывода вполне определенную целевую направленность. Каждый переход оформлялся в виде небольшой программы, анализирующей теорему в контексте заданной ее характеристики. Такие программы, названные приемами программирующего логического вывода, аккумулировались в своеобразном "теоремном" решателе системы. На текущий момент он насчитывает более 1500 приемов.

Прием программирующего вывода — существенно более развитый объект, чем обычное правило вывода в математической логике. Во-первых, он должен самостоятельно находить в базе теорем дополнительные теоремы, которые в сочетании с текущей анализируемой теоремой будут давать полезные следствия. Здесь используются как оглавление базы теорем, так и специальные процедуры быстрого поиска теорем заданного типа. Во-вторых, прием программирующего вывода может обращаться к решателю для различных вспомогательных задач, подсказанных его целевой установокой. В результате срабатывание одного такого приема часто оказывается равносильным длинной цепочке применений обычных правил вывода, причем устройство ее непредсказуемо из-за подключения мощного аппарата всей базы приемов решателя. В-третьих, прием должен использовать определенные эвристические правила для блокировки вывода малополезных (например, чрезмерно громоздких) теорем. В частности, для этого используется блокировка определенных сочетаний последовательно применяемых приемов вывода. При обучении такая блокировка позволила устойчиво обеспечивать исчерпание возможностей дальнейшего вывода в ячейке и выдачу окончательного результата за приемлемое время. В особых случаях результаты вывода выносились в новые ячейки, и глубина вывода таким образом увеличивалась. В-четвертых, прием вывода должен сопровождать теорему характеристиками. Обычно для этого используется общая процедура характеризатора, но иногда прием сам указывает характеристики, объясняющие цель, ради которой он ее получил.

Грань между программирующим логическим выводом и исследовательским выводом является весьма условной. При проработке базы теорем оказалось, что получение многих "классических" теорем может быть объяснено приемами логического вывода того же уровня сложности, что и для их "технических" следствий. Несложные приемы, объясняющие, как одна теорема могла бы быть выведена из других, были созданы, например, для формул корней квадратных и кубических уравнений в элементарной алгебре, теоремы Пифагора и теорем синусов и косинусов в планиметрии, свойств определителей в линейной алгебре, основных формул вычисления первообразных в математическом анализе и т.д. Фактически, теоремы прорабатываются почти подряд, без разделения на

базисные и вторичные. Все это вывело процесс обучения решателей на качественно более высокий уровень. Если раньше анализировались задачи из задачников и нужно было предложить приемы, которые доводили аналогичные задачи до ответа, то теперь анализируются теоремы и предпринимаются попытки создать приемы для "открытия" новых теорем. При этом разрешается использовать весь ранее накопленный системой потенциал решения задач.

Создан прототип генератора приемов, функционирующий по следующей схеме. Выбирается ячейка логического вывода, и в ней запускается процесс вывода теорем. Обычно возникают десятки теорем. В процессе вывода каждая теорема снабжается характеристиками. По этим характеристикам генерируются спецификации приемов (обычно — тоже вплоть до десятка спецификаций на каждую теорему). Компилятор спецификаций преобразует те из спецификаций, для которых пока не созданы приемы, в описания приемов на ГЕНОЛОГе. Для текущего такого приема (пока не откомпилированного на ЛОС) создается тестовый пример. Проверяется, что этот пример не решается системой, после чего новый прием компилируется, и проверяется, что теперь тестовый пример решается. Для отобранных таким образом новых приемов предпринимается расчистка — удаляются приемы, тестовые задачи которых решаются другими новыми приемами. В результате остается лишь малая часть изначально созданных приемов. Для них предпринимается завершающая двухэтапная доводка — сначала происходит прогонка по тому разделу задачника системы, для которого создавались приемы, затем — по всему задачнику. После каждой прогонки отбираются "испортившиеся" задачи, и для восстановления их нормального решения приемы корректируются. В процессе доводки приемы сортируются на пригодные для перенесения в решатель и непригодные. Последние дают информацию для развития генератора приемов. Окончательное перенесение приема в основную базу приемов пока происходит вручную.

Указанный прототип был протестирован на различных разделах базы теорем и позволил создать более 2000 новых приемов, аналогичных ранее созданным вручную и не только им не уступающих, но иногда даже превосходящих, так как при ручном синтезе многие требующие учета особые случаи упускались из виду.

В действительности не все приемы решателя основаны на теоремах. Некоторые из них основаны на тех или иных общих особенностях конкретного раздела, сформулированных в виде так называемых протоколов базы теорем. Эти протоколы уточняют способы алгоритмизации теорем раздела; в частности, порождают приемы, обращающиеся для вывода следствий или преобразований к вспомогательным задачам безотносительно к каким-либо конкретным теоремам. Система, создающая

протоколы при общем рассмотрении раздела базы теорем, названа алгоритмизатором. Технически алгоритмизатор является частью процедуры вывода теорем.

Наконец, упомянем об еще одном классе приемов — общелогических приемах, запрограммированных непосредственно на ЛОСе. Перевод их на ГЕНОЛОГ вряд ли целесообразен, так как ГЕНОЛОГ, по сути дела, является переходником между логическим языком предметной области и языком для описания структур данных, а в указанных приемах языком предметной области как раз и является язык структур данных — ЛОС. Этих приемов немного, они имеют универсальный характер, и автоматизация их создания пока не актуальна. По-видимому, эту автоматизацию можно свести к решению задач, формулируемых в терминах предикатов и операций ЛОСа.

На рисунке 1 приведена диаграмма, на которой представлены основные этапы и основные "действующие лица" изложенного процесса алгоритмизации теорем.

Седьмой [7] и восьмой [8] тома монографии "Компьютерное моделирование логических процессов" посвящены описанию тех блоков диаграммы, которые обеспечивают автоматическое создание приемов. Девятый том, в котором будут представлены алгоритмы программирующего логического вывода, завершит данное описание.

Автор выражает искреннюю благодарность В.Б.Кудрявцеву, поддержка которого сделала возможным проведение данного исследования.

Список литературы

[1] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 1. Архитектура и языки решателя задач*, Физматлит, Москва, 2008, 1024 с.

[2] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 2. Опыт обучения компьютерного решателя задач: логические приемы, алгебра множеств, комбинаторика и элементарная алгебра*, Деп. в ВИНИТИ РАН 09.11.2015, № 184-В2015, Москва, 2015, 1153 с.

[3] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 3. Опыт обучения компьютерного решателя задач: математический анализ, дифференциальные уравнения и элементарная геометрия*, Деп. в ВИНИТИ РАН 09.11.2015, № 185-В2015, Москва, 2015, 1320 с.

[4] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 4. Опыт обучения компьютерного решателя задач: аналитическая геометрия, линейная алгебра, теория вероятностей, комплексный анализ и другие разделы*, Деп. в ВИНИТИ РАН 27.02.2017, № 18-В2017, Москва, 2017, 969 с.

[5] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 5. Опыт обучения компьютерного решателя задач: Элементарные*

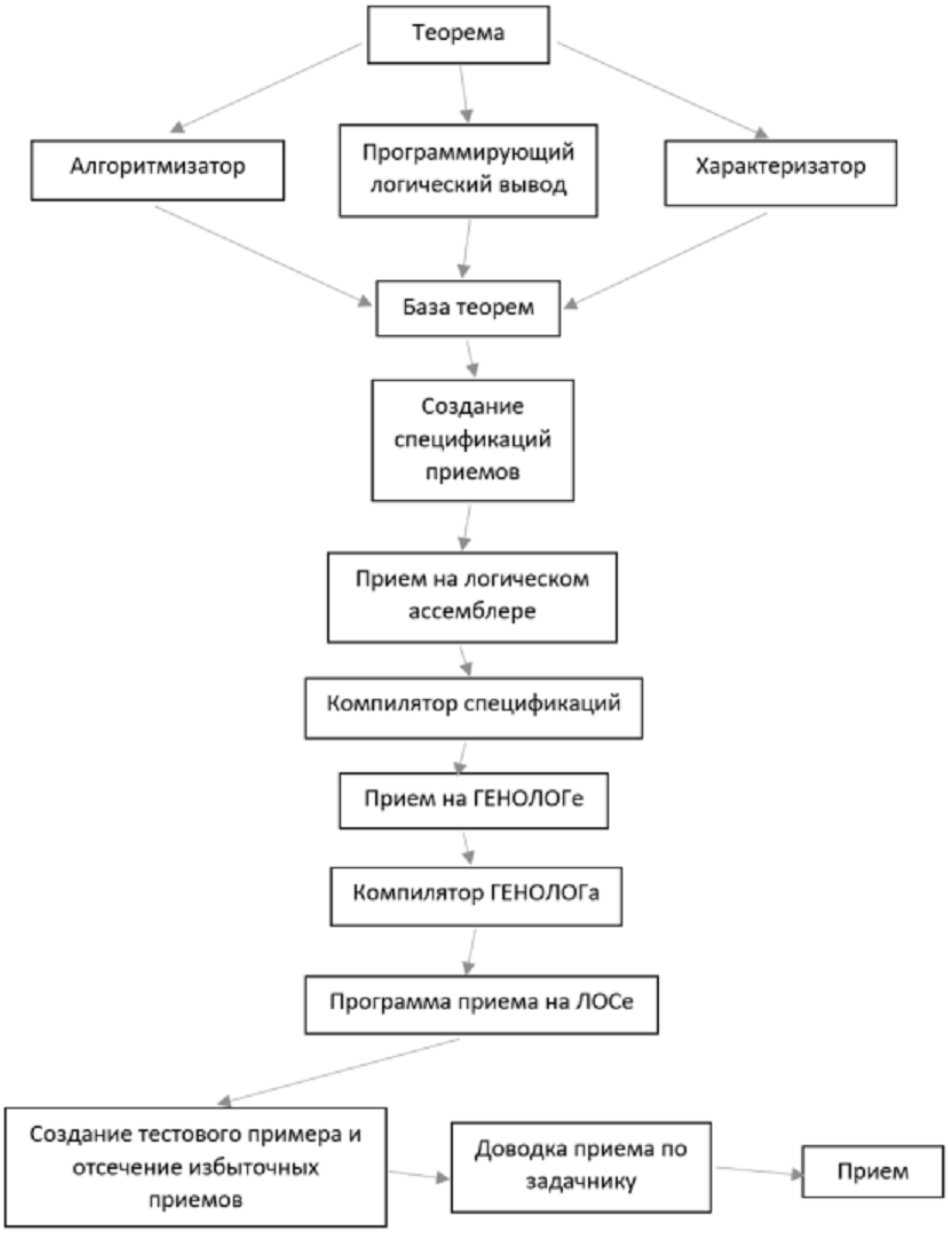

Рис. 1. Процесс алгоритмизации теорем.

физика и химия, шахматы, Деп. в ВИНИТИ РАН 12.08.2019, № 66-В2019, Москва, 2019, 939 с.

[6] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 6. Опыт обучения компьютерного решателя задач: Понимание естественного языка и анализ рисунков*, Деп. в ВИНИТИ РАН 12.08.2019, № 67-В2019, Москва, 2019, 758 с.

[7] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 7. Автоматическое создание приемов логической системы: классификация приемов решателя; логический ассемблер; компилятор спецификаций; создание тестовых примеров и доводка приемов*, Деп. в ВИНИТИ РАН 06.12.2021, № 65-В2021, Москва, 2021, 740 с.

[8] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 8. Автоматическое создание приемов логической системы: база теорем; характеризация теорем; создание спецификаций приемов*, Деп. в ВИНИТИ РАН 06.12.2021, № 66-В2021, Москва, 2021, 516 с.

About algorithmization of knowledge

Podkolzin A.S.

This paper discusses the issues of computer modeling of problem solving processes and automatic creation of solver techniques. The architecture of the process of transitionfrom theorems to techniques (algorithmization of knowledge), which has developed in the process of processing numerous examples, is described. A computer system has been created that allows solving problems from various branches of mathematics and is able to update its database of techniques.

Keywords: computer problem solver, logical system, artificial intelligence.

References

[1] Podkolzin A. S., *Computer modelling of logical processes. Vol. 1. Problem solver architecture and languages*, Fizmatlit, Moscow, 2008 (In Russian), 1024 pp.

[2] Podkolzin A. S., *Computer modelling of logical processes. Vol. 2. Experience in teaching a computer problem solver: logic tricks, set algebra, combinatorics and elementary algebra*, Deposited at VINITI, # 184-B2015, Moscow, 2015 (In Russian), 1153 pp.

[3] Podkolzin A. S., *Computer modelling of logical processes. Vol. 3. Experience in teaching a computer problem solver: mathematical analysis, differential equations and elementary geometry*, Deposited at VINITI, # 185-B2015, Moscow, 2015 (In Russian), 1320 pp.

[4] Podkolzin A. S., *Computer modelling of logical processes. Vol. 4. Experience in teaching a computer problem solver: analytical geometry, linear algebra, probability theory, complex analysis and other sections*, Deposited at VINITI, # 18-B2017, Moscow, 2017 (In Russian), 969 pp.

[5] Podkolzin A. S., *Computer modelling of logical processes. Vol. 5. Experience in teaching a computer problem solver: elementary physics and chemistry, chess*, Deposited at VINITI, # 66-B2019, Moscow, 2019 (In Russian), 939 pp.

[6] Podkolzin A. S., *Computer modelling of logical processes. Vol. 6. Experience in teaching a computer problem solver: natural language understanding and pattern analysis*, Deposited at VINITI, # 67-B2019, Moscow, 2019 (In Russian), 758 pp.

[7] Podkolzin A. S., *Computer modelling of logical processes. Vol. 7. Automatic creation of logic system techniques: classification of solver techniques; logical assembler; compiler of specifications; creating test cases and fine-tuning techniques*, Deposited at VINITI, # 65-B2021, Moscow, 2021 (In Russian), 740 pp.

[8] Podkolzin A. S., *Computer modelling of logical processes. Vol. 8. Automated creation of logical system devices: theorem database, theorem characterization, creation of device specifications*, Deposited at VINITI, # 66-B2021, Moscow, 2021 (In Russian), 516 pp.